Министерство образования Республики Беларусь  
Учреждение образования   
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

По дисциплине «Архитектура вычислительных систем»  
По теме «ОПЕРАЦИЯ УМНОЖЕНИЯ С ФИКСИРОВАННОЙ ТОЧКОЙ»

Выполнил:  
студент гр. 153503  
Киселёва Елизавета Андреевна

Проверил:

Преподаватель

Калиновская Анастасия Александровна

Минск 2023

# 1 Цель работы

Рассмотреть представление чисел в прямом, обратном и дополнительном кодах. Изучить процессы выполнения умножения над целыми числами с фиксированной точкой. Составить программу реализации алгоритма умножения.

**2 Постановка задачи**

***Задание к лабораторной работе 2***

Написать программу эмулятора АЛУ, реализующего операцию умножениянад двумя введенными числами, с возможностью пошагового выполнения алгоритмов.

# 3 Теоретические сведения

**3.1 Прямой код**

Существует несколько соглашений о едином формате представления как положительных, так и отрицательных чисел. Всех их объединяет то, что старший бит слова (с точки зрения европейца — самый левый, или бит, которому при представлении числа без знака должен быть приписан самый большой вес) является битом хранения знака или знаковым разрядом. Все последующие биты слова представляют значащие разряды числа, которые в каждом формате интерпретируются по-своему. Значение 1 в знаковом разряде интерпретируется как представление всем словом отрицательного числа.

|  |  |
| --- | --- |
| 0001001 = | +18 |
| 10010010 = | -18 |

Формат представления чисел в прямом коде неудобен для использования в вычислениях. Во-первых, сложение и вычитание положительных и отрицательных чисел выполняется по-разному, а потому требуется анализировать знаковые разряды операндов. Во-вторых, в прямом коде числу 0 соответствуют две кодовых комбинации:

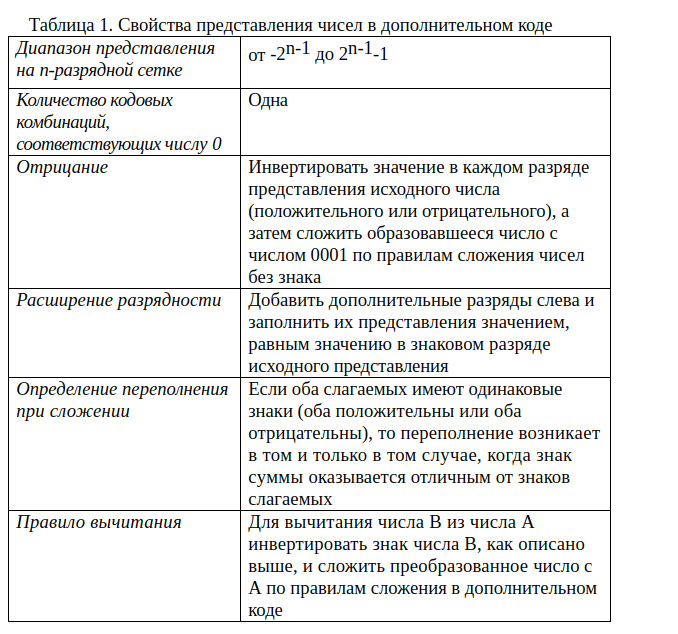
|  |  |
| --- | --- |
| 0000000= | +010 |
| 10000000= | -010 |
|  |  |

Это также неудобно, поскольку усложняется анализ результата на равенство нулю, а такая операция в программах встречается очень часто.

Из-за этих недостатков прямой код практически не применяется при peaлизации в АЛУ арифметических операций над целыми числами. Вместо этого более широкое применение находит другой формат, получивший наименование дополнительного кода.

## **3.2 Дополнительный код**

Как и в прямом, в дополнительном коде старший разряд в разрядной сетке отводится для представления знака числа. Остальные разряды интерпретируются не так, как в прямом коде. В табл. 1 перечислены основные свойства дополнительного кода и правила выполнения арифметических операций в дополнительном коде, которые мы рассмотрим в этом и следующем разделах.



В большинстве описаний дополнительного кода основное внимание уделяется технике формирования представления отрицательного числа по представлению соответствующего положительного, причем не приводится формальное доказательство работоспособности описанной схемы. Мы решили нарушить эту традицию, и в данном разделе, а также в следующем будем основываться на описании, в котором это представление рассматривается в терминах взвешенной суммы значений разрядов. Такой способ мы уже использовали выше при описании представления чисел без знака и целых чисел со знаком в прямом коде. Преимущество такой методики в том, что она не оставляет ни малейших сомнений в справедливости излагаемых правил выполнения арифметических операций в любых частных случаях.

## **3.3 Умножение**

Алгоритмы выполнения умножения значительно сложнее, чем сложения или вычитания, причем в современных вычислительных системах можно встретить как аппаратную его реализацию, так и программную. В данном разделе мы ставили перед собой задачу дать читателю общее представление о подходе, на основе которого такие алгоритмы пpoeктируются. Начнем с простой задачи перемножения двух чисел без знака (т.е. неотрицательных чисел), а затем рассмотрим один из наиболее широко известных алгоритмов умножения целых чисел со знаком, представленных в двоичном коде.

Ниже показана схема выполнения умножения двоичных чисел без знака, знакомая всем в десятичной интерпретации ещё со школьной скамьи, которую ещё называют умножением в столбик.

|  |  |
| --- | --- |
| *1011* | *Множимое(11)* |
| *х* *1101* | *Множитель(13)* |
| *1011* |  |
| *0000* | *Частичные произведения* |
| *1011* |  |
| *1011* |  |
| *1000111* | *Произведение(143)* |

Анализируя этот пример, отметим следующее.

1. При выполнении умножения необходимо формировать частичные произведения, по одному на каждый разряд множителя. Эти частичные произведения затем суммируются, а их сумма и есть результат умножения — полное произведение.

2. Сформировать частичные произведения в двоичном коде довольно легко. Если соответствующий разряд множителя равен 0, частичное произведение также равно 00..00. Если соответствующий разряд множителя равен 1 частичное произведение равно множимому.

3. Полное произведение вычисляется суммированием частичных произведений, причем каждое очередное частичное произведение в этой сумме сдвигается на одну позицию влево относительно предыдущего.

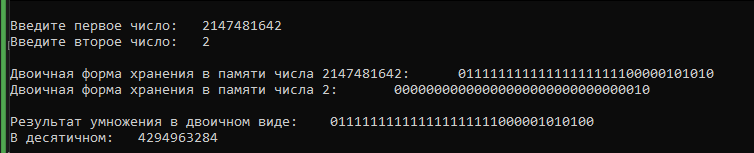
4. Результатом перемножения двух n-разрядных целых чисел будет 2n–разрядное число.

Реализация алгоритма умножения техническими или программными средствами позволяет несколько повысить его эффективность по сравнению с тем вариантом, который мы традиционно используем при вычислении в столбик вручную. Во-первых, суммирование очередного частичного произведения можно выполнять немедленно после того, как оно будет сформировано, не дожидаясь остальных. Таким образом, отпадает необходимость в средствах для временного хранения частичных произведений, т.е. для аппаратной реализации потребуется меньше регистров. Во-вторых, можно сберечь время, необходимое для формирования частичных произведений. Для каждого разряда в коде множителя, равного 1, необходимо выполнить сдвиг и сложение кода множимого, а для разряда, равного 0, — только сдвиг.

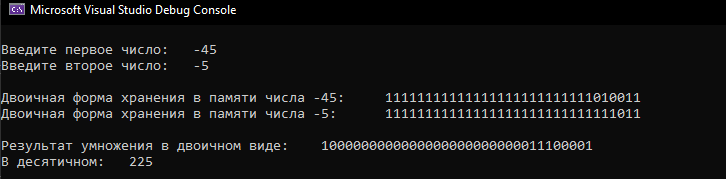
# 4 Пример работы программы

# 

# Тестовый пример 1



# Тестовый пример 2



# Тестовый пример 3

# 5 Выводы

В ходе выполнения лабораторной работы была написана программа, реализующая операции умножения над числами с фиксированной точкой в дополнительном коде.

# Приложение 1. Текст программы

#include <iostream>

#include <regex>

#include <functional>

#include <string>

typedef long long ll;

int number1[33];

int number2[33];

ll num1\_dec;

ll num2\_dec;

template<typename ReturnValue>

ReturnValue CorrectInput(const std::string& variable\_name, const std::regex& pattern,

std::function<ReturnValue(std::string& input)> converter) {

std::string value;

do {

std::cout << "Введите " << variable\_name << ":\t";

std::cin >> value;

} while (!std::regex\_match(value, pattern));

return converter(value);

}

void to\_binary(int mask[], ll n) {

int j = 31;

while (n > 0) {

mask[j--] = n % 2;

n >>= 1;

}

}

void to\_additional(int arr[]) {

for (int i = 0; i < 32; ++i)

arr[i] = (arr[i] + 1) % 2;

int j = 31;

arr[j]++;

while (arr[j] == 2) {

arr[j--] %= 2;

arr[j]++;

}

}

ll to\_dec(std::string& s) {

if (num1\_dec < 0 || num2\_dec < 0)

return num1\_dec \* num2\_dec;

ll res = 0;

for (std::string::size\_type i = 0; i < s.size(); i++) {

res <<= 1;

res += s[i] - '0';

}

return res;

}

std::string mul(const std::string& a, const std::string& b) {

std::string res(a.length() + b.length() - 1, '0');

for (int i = a.length() - 1; i >= 0; i--)

for (int j = b.length() - 1; j >= 0; j--)

res[i + j] += (a[i] != '0' && b[j] != '0');

for (std::string::size\_type i = res.length() - 1; i > 0; i--) {

res[i - 1] += (res[i] - '0') / 2;

res[i] = (res[i] - '0') % 2 + '0';

}

while (res[0] > '1') {

res = "0" + res;

res[0] += (res[1] - '0') / 2;

res[1] = (res[1] - '0') % 2 + '0';

}

return res;

}

void print\_array(int arr[]) {

for (int i = 0; i < 32; ++i)

std::cout << arr[i];

}

std::string from\_int\_to\_str(int a[]) {

std::string res;

for (int i = 0; i < 32; ++i)

res.push\_back(a[i] ? '1' : '0');

return res;

}

int main() {

setlocale(LC\_ALL, "rus");

std::cout << '\n';

num1\_dec = CorrectInput<ll>(std::string("первое число"), std::regex("-?[0-9]\*"), [](std::string& s) {return atoi(s.c\_str()); });

num2\_dec = CorrectInput<ll>(std::string("второе число"), std::regex("-?[0-9]\*"), [](std::string& s) {return atoi(s.c\_str()); });

to\_binary(number1, abs(num1\_dec));

to\_binary(number2, abs(num2\_dec));

if (num1\_dec < 0)

to\_additional(number1);

if (num2\_dec < 0)

to\_additional(number2);

std::cout << "\nДвоичная форма хранения в памяти числа " << num1\_dec << ":\t";

print\_array(number1);

std::cout << "\nДвоичная форма хранения в памяти числа " << num2\_dec << ":\t";

print\_array(number2);

std::string binary\_res = mul(from\_int\_to\_str(number1), from\_int\_to\_str(number2));

binary\_res.erase(binary\_res.begin(), binary\_res.begin() + 30);

std::cout << "\n\nРезультат умножения в двоичном виде:\t" << binary\_res;

std::cout << "\nВ десятичном:\t" << to\_dec(binary\_res) << '\n';

return 0;

}